流量感知的可重构路由算法

付斌章 韩银和 李华伟 李晓维

摘 要：在众核处理器系统中，片上网络常被用来提供高带宽、低延迟、高可靠的片上网络通信。为了减少网络拥塞、提高网络性能，流量平衡路由算法获得研究人员的广泛关注。流量平衡算法通常利用完全自适应路由算法来提供路径分集，而当前的完全自适应路由算法或者需要较多的虚通道或者假设一个保守的流控策略。一方面虚通道是比较昂贵的资源，另一方面保守的流控策略则有可能造成网络性能的下降。因此研究人员提出利用应用程序的流量信息来提升路由性能。这些算法在不使用虚通道的基础上可以针对不同的流量特性进行重构，从而实现路由自适应度的按需分配。按照使用的流量信息类型，流量感知的可重构路由算法可以分为离线和在线算法。离线算法需要事先知道程序的流量特征，因此他们大多针对应用程序定制的多核片上系统。在线算法则是根据在线收集的流量信息进行重构，因此可以用于通用处理器系统。本文将讨论最近国际上提出的两种著名的离线算法，并重点介绍本文作者在 2011 年国际计算机体系结构大会(ISCA'11)上发表的基于算盘转向模型的在线可重构路由算法。

关键词：片上网络，路由算法，路由重构，流量平衡；

1 引言

由于存储器墙、ILP[1]墙、以及功耗墙等约束的存在，传统的依靠提升单个处理器性能的做法已不能获得令人满意的结果。此时多核及众核处理器通过并行化来提升应用程序性能的做法被认为是一种可行方案[1]。在众核处理器系统中，片上网络由于具有优良的性能，所以被广泛期望成为主流互连方案[2]。由于片上网络负责为处理器之间或处理器和缓存之间提供通信，所以其性能对系统性能具有很大影响。一般来说，片上网络的性能主要取决于网络拓扑、流控策略和路由算法。

网络拓扑决定了任意两点之间的最短距离以及网络的对剖带宽，从而决定了网络的峰值性能。网络拓扑的设计需要综合考虑各种因素，例如端口数目、每个端口的带宽、工艺所能允许的布线密度以及信号速率。目前常见的网络拓扑包括交叉开关矩阵，克洛斯网络（Clos Network）[2]，蝴蝶网络，以及 Tori 网络（包括网状网（mesh），环网（torus），以及超立方（hypercube）等）[3]。其中二维网状网的扁平结构便于生产加工，所以被片上网络系统广泛采用[4][5][6]。本文将主要针对二维网状网网络进行讨论。

流控机制负责为数据包分配网络资源，例如通道带宽、缓存空间、以及状态[3]。好的流控机制应该能够准确、高效地将网络资源分配给最需要该资源的数据包。在包交换网络中常见的流控机制包括存储转发机制、虚跨步机制、虫孔机制以及虚通道机制。由于虫孔机制可以有效地减少缓存需求及数据包延迟，所以被片上网络系统广泛采用。虫孔机制将数据包切分为多个流控单元（flow control digit，简称 flit）。其中每个数据包包括一个头流控单元、若干数据流控单元、以及一个尾流控单元。头流控单元用于建立通路，后续的流控单元都将沿着同样的路径前进，最终尾流控单元释放该通路。本文的讨论基于采用虫孔机制的片上网络。

路由算法用于决定数据包在网络中的传输路径。好的路由算法应该能够为数据包指定一

[1] Instruction-Level-Parallelism，指令级并行
[2] 一类多级交换网络的拓扑，由查尔斯.克洛斯（Charles Clos）于 1953 年提出

条路径使其最快地到达目的节点[3]。根据数据包在网络中的传输距离，路由算法可以分为最短路径路由算法和非最短路径路由算法。最短路径路由算法要求数据包的每一跳均选择趋近目标节点的方向。这种算法的好处是规则简单，并且网络中不会产生活锁。相对来说，非最短路径路由算法的规则较为复杂，但是可以提升网络对拥塞以及故障的容忍能力。为了便于理解，本文讨论主要针对最短路径路由算法。

根据在路由过程中是否考虑网络状态，路由算法可以分为无关路由算法（oblivious routing）以及自适应路由算法（adaptive routing）。无关路由不考虑网络状态，所以算法相对简单、便于实现。但是其对网络拥塞、路由器以及链路的故障的容忍能力较低。自适应路由算法可以根据网络的当前状态动态地选择路径，从而避免拥塞。

按照可使用的路径数量，自适应路由算法可以分为部分自适应路由算法和完全自适应路由算法。完全自适应路由算法允许数据包利用源和目的节点之间的任意路径，而部分自适应路由算法则只允许数据包利用其中一部分路径。

由于需要足够的路径分集来缓解拥塞，目前流量平衡路由算法大多利用一个完全自适应路由算法来提供被选输出端口[7][8][9][10][11][12]。然而目前常见的完全自适应路由算法不是需要大量的虚通道[13][14]，就是要假设一个非常保守的流控策略[15][16][17]。

由于在片上网络系统中虚通道的实现成本相对较高，所以这些需要大量虚通道的路由算法，例如引文[13][14]中的方案，并不适用于片上网络系统。基于杜亚托（Duato）理论的路由算法对虚通道的要求较低，但是要求保守的流控策略。根据杜亚托理论的要求，路由器中的缓冲队列不允许存放属于不同数据包的流控单元[15]。只有当上一个数据包的尾流控单元离开之后，缓冲队列才能被重新分配。这个约束条件保证了当某个数据包发生拥塞时，其头流控单元一定在缓冲队列的头部，从而可以选择逃避通道。然而这却造成缓冲空间的浪费。因此对于传输短数据包的网络，这种约束条件将会导致网络性能的下降[18]。

与完全自适应路由算法相比，不使用虚通道的部分自适应路由算法的实现开销更小并且可以使用更加先进的流控策略。例如格拉斯（C. J. Glass）和倪明选（L. M. Ni）提出的转向模型[19]和邱（音译，G. M. Chiu）提出的奇偶转向模型[20]。转向模型可用于设计无虚通道的部分自适应路由算法。他们将网络中所有可能的转向划分为两种抽象环：顺时针抽象环和逆时针抽象环。通过在每个抽象环中禁止一个转向，可以保证网络不出现死锁。然而邱发现根据转向模型设计的路由算法所提供的自适应度是不均匀的。为了解决这个问题，他提出了奇偶转向模型。但是奇偶转向模型对于任何距离大于 2 的节点对均不能提供全部的自适应度。不均匀或不充分的路由自适应度可能导致在突发流量情况下的网络拥塞。

为了解决这个问题，研究人员提出使用可重构的路由算法为应用程序提供按需分配的路由自适应度。总体来说可重构路由算法是部分自适应路由算法，所以可以不使用虚通道以及不要求保守的流控策略。这些算法可以通过分析应用程序的流量特性为高负载方向的数据包提供更多的路径分集。

根据流量信息的获取方式，可重构路由算法可以分为离线和在线算法两类。离线可重构路由算法假设可以提前获得应用程序的流量信息，并且通过对流量特性的分析来决定路由策略。具有代表性的离线可重构路由算法包括发表在 2009 年 IEEE 期刊 TPDS 第 3 期的应用程序定制的路由算法[21]和发表在 2009 年国际计算机体系结构大会(ISCA’09)上的应用程序感知的无关路由算法[22]。

相对于离线算法，在线可重构路由算法可以根据在线收集的流量信息对自身重构。因此

在线可重构路由算法可以应用于通用处理器系统。本文将介绍计算机体系结构国家重点实验室发表在 2011 年国际计算机体系结构大会(ISCA'11)上的基于算盘转向模型的可重构路由算法[23]。

2 离线的可重构路由算法

设计离线可重构路由算法的核心问题是如何在保证路由算法无死锁的基础上使网络的性能得到最大提高。大多数离线算法的解决思路是首先构建通道依赖图，通过在通道依赖图中去除所有环的方式来保证路由算法的无死锁，并通过在各个通道中平衡流量的方式来提高网络性能。本文所要讨论的两个离线算法均是遵循这个基本思路，但是具有不同的侧重点。

2.1 按应用定制的路由算法[21]

按应用定制的路由算法（Application Specific Routing Algorithms，APSRA）利用程序中某些节点可能不通信或者某些节点对之间不同时通信的特点来提高路由算法的性能。首先按应用定制的路由算法将应用程序抽象为任务图，并将任务映射到不同的处理器节点。然后通过参考网络的拓扑图，将任务图转化为通道依赖图。最后分析通道依赖图，并通过确保其无环的方式保证路由算法的无死锁。另外在去环的过程中，通过均衡各个通道内流量的方式来提升路由算法的性能。我们将使用文献[21]中的一个例子来介绍其基本的工作原理。具体的算法和实现细节请参考文献[21]。

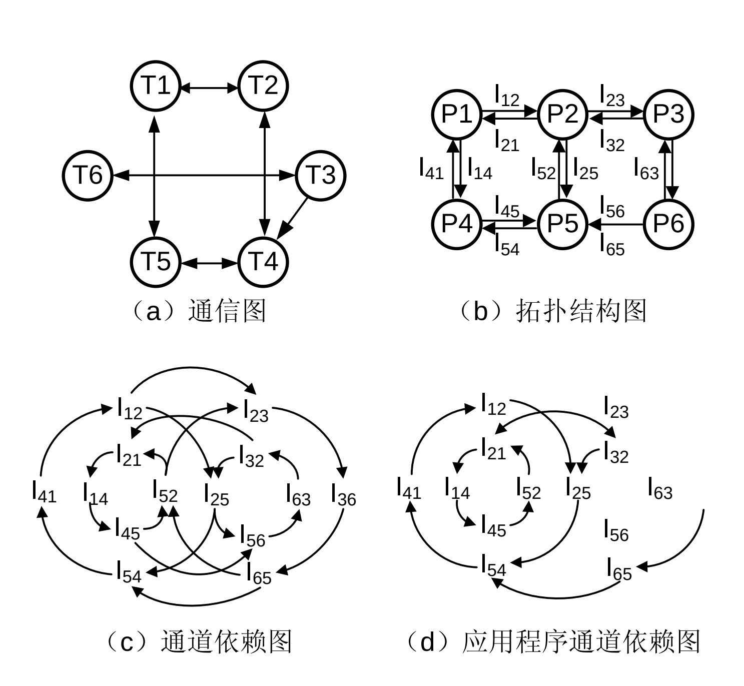


图1. 按应用定制的路由路由算法示例

如图 1(a)所示，通信图表明该应用程序包含 6 个任务。任务之间的通信用箭头表示，其中双箭头表示两个任务可以相互通信，而单箭头则表示通信是单方向的。图 1 (b)给出了网络的拓扑结构，其中圆圈代表处理器，箭头代表连接处理器的通道。在这个例子中，假设任务 T_i 被映射到处理器 P_i，即 $M(T_i)=P_i$, i=1, 2, …, 6，其中 M 为映射函数。为了生成通道依赖图，首先假设路由算法为完全自适应最短路径路由算法。此时生成的通道依赖图如图 1 (c)所示。由于在通道依赖图中包含 6 个依赖环，所以根据达利（Dally）提出的理论[14]网络不能使用完全自适应路由算法[3]。然而事实上并不是所有的任务之间都有通信，因此在去掉不存在的通道依赖之后的应用程序通道依赖图如图 1 (d)所示。例如依赖 $l_{12}\rightarrow l_{23}$ 在通道依赖图图 1 (c)）中是存在的，但是在应用程序通道依赖图图 1 (d)）中是不存在的。通过分析拓扑结构图，可以发现只有通信 $T_1\rightarrow T_3$，$T_1\rightarrow T_6$ 和 $T_4\rightarrow T_3$ 可以引入依赖 $l_{12}\rightarrow l_{23}$。但是根据通信图，这三种通信都是不存在的。因此依赖 $l_{12}\rightarrow l_{23}$ 事实上是不会出现的，所以可以在应用程

[3] 此处文献[21]讲述的是“根据 Duato 理论[15]不能使用完全自适应路由算法”。然而文献[21]的这种表述是错误的，因为 Duato 理论是允许通道依赖图中包含环的[15]。所以本文将其修改为“根据 Dally 理论[14]”。

序通道依赖图中去除。最后生成的应用程序通道依赖图包含两个环，因此根据达利的理论仍然不能使用完全自适应路由算法。为了去除环，l_{41}→l_{21} 的依赖和 l_{14}→l_{45} 的依赖被禁止。

此时虽然不会出现死锁，但是路由的自适应度受到了损失。如果可以事先知道任务间通信发生的时间，例如 T_1→T_5 的通信和 T_2→T_4 的通信在时间上不重叠，虽然应用程序通道依赖图中存在环，但事实上这个环永远不会在网络中形成。所以我们不需要去除这些环，从而生成的路由算法仍然是完全自适应路由算法。

2.2 应用程序感知的无关路由算法[22]

与按应用定制的路由算法不同，文献[22]认为由于自适应路由算法会增加路由算法的实现复杂度所以应该使用无关路由算法。为了针对不同应用程序的特性，文献[22]提出一个应用程序感知的无关路由生成流程。这个流程主要包含以下五个步骤：

1. 构造无环的通道依赖图；
2. 根据应用程序的通信图构建流图；
3. 在流图中为每个通信流选择通路；
4. 如果满足要求，返回步骤 1；
5. 选择最好的路径集合。

步骤 1：通过在通道依赖图中去除所有的环可以保证生成的无关路由算法不会产生死锁。事实上在一个有向图中搜索和去除所有的环是一个比较复杂的工作，因此文献[22]提出使用转向模型[19]来构造无环的通道依赖图。例如首先假设采用的路由算法为北最后（north-last）路由[19]，此时生成的通道依赖图一定是无环的。构造通道依赖图的方法读者可以参考文献[14]。例如针对如图 2 (a)所示的 3×3 网状网网络，其对应于“北最后”路由算法的通道依赖图如图 2 (b)所示。当使用北最后路由将数据包从节点 E 向节点 G 路由时，数据包可以沿着通路 E→D→G 前进。所以通道 ED 依赖于通道 DG。相应的在图 2 (b)中存在一条 ED 到 DG 的边。显然，图 2 (b)中不存在环。

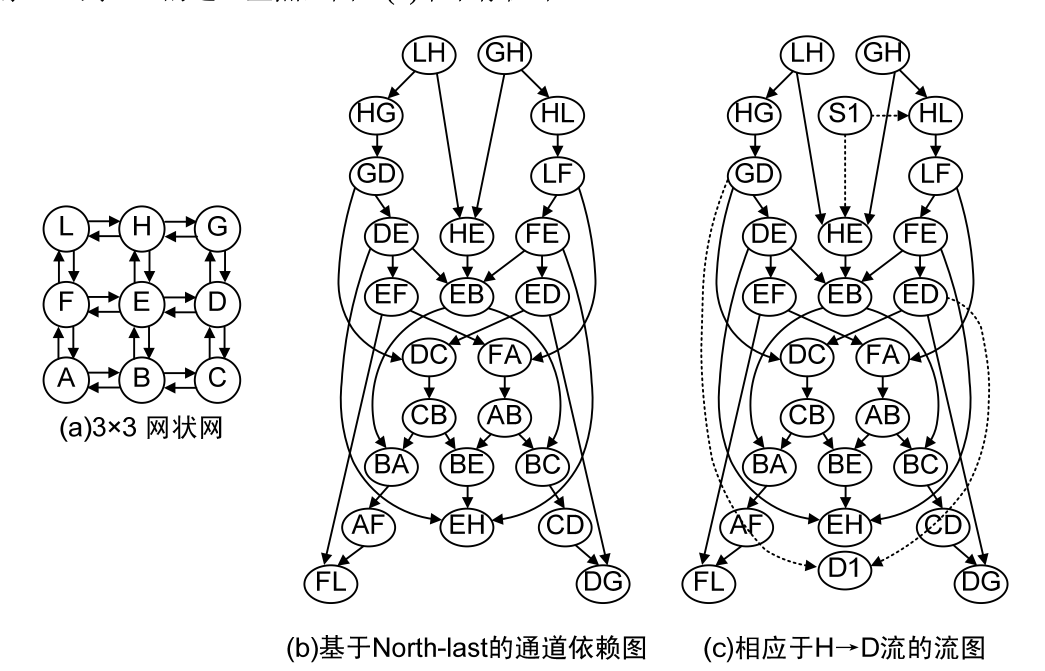


图2. 通道依赖图和流图构建示例

步骤 2：是在步骤 1 的基础上，通过在无环的通道依赖图中插入“哑节点”的方式构建

流图。所谓哑节点实际上是网络中的一个路由器。例如应用程序存在从节点 H 到节点 D 的通信流。那么为了构建流图，首先将节点 H 和节点 D 当作两个哑节点插入图中。然后将源节点 H（标注为 S_1）与其所有的输出通道相连接，并将目的节点 D(标注为 D_1)与其所有的输入通道相连接。如图 2 (c)所示，与哑节点相连的边用虚线表示。图 2 (c)只添加了一个通信流 H→D，实际上应用程序中可能包含若干流，最终的流图应该包含所有的通信流。

步骤 3：在构造好的流图中为每个通信流选择一条路径。在选择路径的过程中，路径的分配应该满足一些约束条件，例如使最大通道负载最小。文献[22]针对小规模网络将路径的选择问题抽象为混合的线性规划问题；针对大规模的网络，则提出一个贪婪算法来选择路径。首先将通信流按照对网络带宽的需求进行降序排列，然后按顺序针对每个通信流为其选择路径。选择路径的方式是遵循加权的迪杰斯特拉（Dijkstra）最短路径算法。

步骤 4：检查是否所有的通信流都被分配了路径，以及网络状态是否满足优化的目标。如果满足则返回步骤 1。从而使用另外一个转向模型路由算法来构建新的无环的通道依赖图，并且为每个通信流选择通路。

步骤 5：比较针对不同转向模型路由算法生成的通信流路径，并根据事先定义的标准选择一个最优的路径集合。

相应的路径集合则通过写路由表的方式写入芯片，从而应用程序可以按照优化后的路径路由通信流。文献[22]还讨论了当网络中存在多条虚通道情况下的优化方法。例如可以在不同的虚通道内使用不同的转向模型路由算法来构建不同的流图，从而将不同的通信流划分到不同的虚拟网络内，以更好地平衡网络流量。详细内容可参阅文献[22]。

3 在线可重构路由算法

本节将讨论我们在文献[23]中提出的算盘转向模型及相应的可重构路由算法。

3.1 算盘转向模型基本想法

原始的转向模型已经证明如果网络中所有允许的转向不会构成抽象环，那么网络无死锁[19]。文献[20]进一步证明如果顺时针和逆时针抽象环中的最右边都不存在，那么网络无死锁。如图 3 (a)所示，顺时针最右边由两个转向（东向南（ES）和南向西（SW）转向）以及若干北向南（NS）通道组成。为了在网络中去除所有最右边，文献[20]要求所有在奇数列的节点禁止南向西转向（如图 3 (b)所示），并要求所有在偶数列的节点禁止东向南转向（如图 3 (c)所示）。禁止逆时针最右边的原理类似，本文省略了具体的讨论。

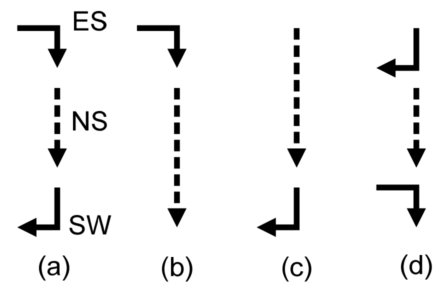


图3. 顺时针最右边的构成

(a) 东向南转向在南向西转向上方；(b) 禁止南向西转向；(c) 禁止东向南转向；(d) 东向南转向在南向西转向下方。图中虚线表示连续的北向南通道

文献[23]继承了文献[20]的核心思想，即“如果网络不出现顺时针和逆时针最右边，那么该网络无死锁”。与原始的转向模型不同的是，文献[23]去除最右边的方法更灵活。如图 3 (a)所示，为了构成顺时针最右边，必须存在一个东向南转向在一个南向西转向上方。因此可以通过在所有的南向西转向上方禁止东向南转向的方式来去除顺时针最右边（如图 3 (d)所示）。采用这种方式后网络中的每一列必然存在一个点；在这个点的上方所有的东向南转向被禁止，而在这个点的下方所有的南向西转向被禁

止。文献[23]将这种类型的节点称为顺时针珠子节点。类似的，在网络的每一个列同样存在一个逆时针的珠子节点用于分隔逆时针转向。因此，也不会出现逆时针最右边。

3.2 算盘转向模型

如图 4 (a)和图 4 (b)所示，一个 4×4 网状网网络被比喻为一个算盘，每一列假设存在一个滑竿以及两个滑动珠子：顺时针珠子和逆时针珠子。图中虚线边的矩形块代表珠子节点可能存在的位置，实心椭圆代表顺时针珠子，空心椭圆代表逆时针珠子，虚线箭头代表禁止的转向。为了图的清晰性，允许的转向并没有示出。顺时针珠子和逆时针珠子可以独立控制，并且被分别用来确定每一列中顺时针转向和逆时针转向的分布。

算盘转向模型可以定义为如下三个规则：

1. 所有顺时针（相应的，逆时针）珠子节点上方的节点禁止东向南（相应的，北向西）转向，
2. 所有顺时针（相应的，逆时针）珠子节点下方的节点禁止南向西（相应的，东向北）转向，
3. 顺时针（相应的，逆时针）珠子节点不禁止顺时针（相应的，逆时针）转向。

图 4 (c)和图 4 (d)展示了遵循算盘转向模型的禁止转向的分布情况。根据算盘转向模型，顺时针（相应的，逆时针）珠子节点的持有者可以允许所有的顺时针（相应的，逆时针）转向。在顺时针（相应的，逆时针）珠子节点上方的所有路由器允许除了东向南转向外所有的顺时针转向（相应的，除北向西转向外所有的逆时针转向），而在其下方的路由器则允许除了南向西转向外所有的顺时针转向（除东向北转向外所有的逆时针转向）。按照这种规则分布的转向不会形成顺时针和逆时针最右边。因此根据算盘转向模型，设计无死锁路由算法便简化为在网络的每一列中确定顺时针和逆时针珠子节点的位置。一旦位置确定，便形成新的路由规则。

对于一个 k×k 网状网网络，每一列包含两个珠子节点。因为每个珠子节点有 k 个可能的位置，因此在一列中，会具有 k×k 个配置情况。而对于一个网络来说，因为具有 k 个列，所以有$(k×k)^k$ 种配置情况。每一种珠子节点的配置方式代表一种路由的配置，因此路由算法的重构此时转化为如何在网络的每一列中移动珠子。下面通过一个例子来介绍基于算盘转向模型的可重构路由算法的工作原理。

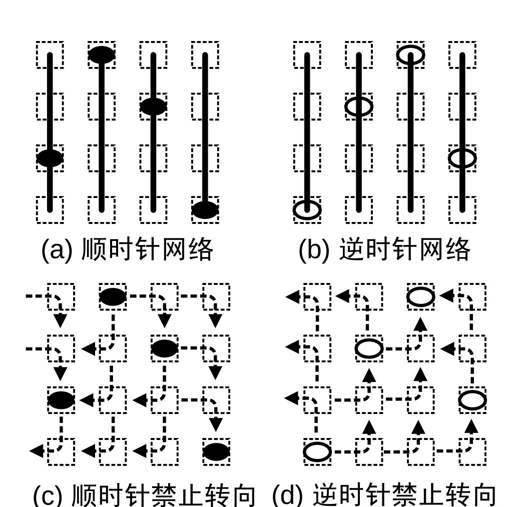


图4. 应用算法转向模型的网络

如 5 (a)所示，初始化时将顺时针珠子放在网络的最低一行。因此根据算盘转向模型，珠子节点上方的节点全部禁止东向南转向。此时假设检测到一个热点节点 5，并且节点 6 需要在一个相对比较长的时间内向其发送数据包。因为对这组节点而言只有一条可使用的最短路径，所以该路径非常容易产生拥塞。节点 6 为了获得更多可使用的通路来平衡流量，向其东侧邻居节点 7 抱怨当前状况，即节点 7 禁止了节点 6 需使用的东向南转向。节点 7 收集来自其邻居的抱怨，并与珠子的拥有者节点 1 进行协商。珠子的拥有者会评估来自不同节点的请求，最终决定是否放弃珠子的所有权以及需要将珠子传递给谁。在当前例子中，节点 1 会将珠子传递给节点 7。如图 5 (b)所示，

节点 7 获得顺时针珠子之后，可以允许东向南转向。此时节点 6 到节点 5 便有两条通路可用来平衡流量。与节点 6 类似，节点 7 也会向其邻居节点 8 抱怨，因为节点 8 同样禁止东向南转向。经过类似的过程之后，在当前例子中，节点 8 同样会获得顺时针珠子的拥有权限，并且打开东向南转向。此时如图 5 (c)所示，节点 6 到节点 5 之间的所有最短路径均可以提供给数据包用于流量平衡。

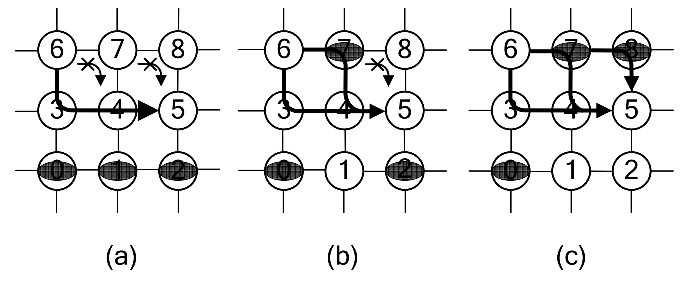


图5.　算盘转向模型路由示例

3.3 算盘珠安全传递

为了保证网络的无死锁特性，在网络中移动珠子节点同样是一件很有挑战的工作。如图 6 (a)所示，节点 1 是顺时针珠子的拥有者，所以其允许南向西转向。此时节点 4 可以利用这个南向西转向向节点 0 发送数据包。如果此时珠子节点被向上移动（如图 6 (b)所示），节点 1 将禁止南向西转向。如果节点 4 并没有及时发现这个改变并继续向节点 1 发送目的地为节点 0 的数据包，那么这些数据包将被阻塞在节点 1。另外节点 7 获得顺时针珠子之后，可以打开东向南转向。然而如果这个转向打开的时间过早，例如节点 1 和节点 4 中仍然有数据包在利用南向西转向，那么网络可能会出现顺时针最右边从而违反算盘转向模型的规则。

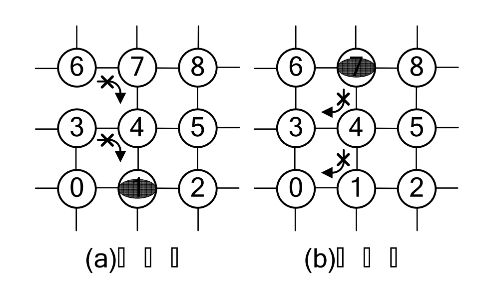


图6.　珠子移动前后禁止转向分布

为了解决这些问题，珠子的移动需要满足下面两条规则：

1. 如果确信没有数据包正在使用或将要请求使用某一转向，则允许该转向被禁止。
2. 如果确信没有数据包在使用任何可以与某一转向构成最右边的转向，则该转向允许被打开。

一般来说，为了将珠子移动 h 跳距离，分别需要打开和禁止 h 个转向。因此对于一个大规模网络而言，同时满足上面两个要求是一件非常难的工作。为了简化任务复杂度，文献[23]将珠子的移动切分成 h 个子步骤，在每一步中珠子节点只被移动一跳距离。这个基本的子步骤被视为一个安全的原子操作并被命名为``珠子传递"（bead-passing）。使用这个安全的原子操作具有两个好处：首先珠子传递可以在局部范围内通过邻居之间的交互完成，因此具有良好的可扩展性；另外珠子传递本身可以保证网络在重构过程中不会引入死锁，因此设计人员在设计自己的可重构路由算法时不需要考虑复杂的死锁问题。

在每一次原子操作中，只有当前的珠子拥有者需要禁止一个转向。例如在图 5 所示的例子中，为了将珠子从节点 1 上移至节点 4，节点 1 需要禁止南向西转向。为了做到这一点，节点 1 首先需要通知节点 4 禁止向其发送目的地在西南方向的数据包，因为这些数据包可能需要使用南向西转向。当节点 4 收到这个通知时，它将自身的南输出端口设置为“不接收西南方向数据包”。此后所有西南方向的数据包在节点 4 内都会通过西输出端口被路由到节点 3。但是此时节点 4 并不能向节点 1 发送确认，因为在这之前可能存在已经完成路由阶段的数据包。而这些数据包在路由时，并不知道节点 1 会禁止南向西转向。因此在向节点 1 发送确认之前，节点 4 还需要在自身内部排空可能发往节点 1 的西南数据包。

```
1.  IF 向上移动 Then
2.      通知北边邻居从本地、东和北输入缓冲队列清空西南方向数据包。
3.      等待北边邻居完成。
4.      在北输入缓冲队列中清空西南方向数据包。
5.      禁止南向西转向,将珠子向上传递。
6.  ELSE IF 向下移动 Then
7.      通知西边邻居从本地、西和北输入缓冲队列清空东南方向数据包
8.      等待西边邻居完成。
9.      在西输入缓冲队列中清空东南方向数据包。
10.     禁止东向南转向，将珠子向下传递。
11. ENDIF
```

（a）移动顺时针珠子

```
1.  IF 向上移动 Then
2.      通知西边邻居从本地、西和南入缓冲队列清空东北方向数据包。
3.      等待西边邻居完成。
4.      在西边输入缓冲队列中清空东北方向数据包。
5.      禁止东向北转向，将珠子向上传递。
6.  ELSE IF 向下移动 Then
7.      通知南边邻居从本地、东和南输入缓冲队列清空西北方向数据包。
8.      等待南边邻居完成。
9.      在南输入缓冲队列中清空东南方向数据包
10.     禁止北转西转向，将珠子向下传递。
11. ENDIF
```

（b）移动逆时针珠子

图7.　珠子传递的伪代码

路由器内部清除某种类型的数据包的方法在路由重构领域已经获得广泛的研究[24][25][26][27][28][29]。文献[23]借鉴文献[29] 重构令牌的思想，但是文献[23]方法的不同之处在于并不是所有的数据包都需要排空。在上述例子中节点 4 只需要排空西南方向的数据包。因为节点 4 只有可能从本地、东和北输入端口接收到西南数据包，重构令牌只需要在相应的输入缓冲队列的末尾插入即可。当南输出端口收到所有来自这三个输入端口的令牌之后，节点 4 便可以确认在其内部不再存在可能被路由到节点 1 的西南方向数据包。因此节点 4 此时可以向节点 1 发送确认。

收到确认之后，节点 1 便有足够的信心认为其北输入端口不会再接收到新的西南方向数据包。但是此时在北输入端口的缓冲队列中仍可能存在西南方向数据包。因此在禁止南向西转向之前，节点 1 还需要在其自身的北输入端口排空西南方向数据包。在此之后，节点 1 可以禁止南向西转向并将珠子向上传递。接收到珠子后，节点 4 打开东向南转向并通知其西边的邻居可以向其发送东南方向数据包。到此完成一次珠子传递。如图 7 所示，移动顺时针和逆时针珠子的操作被总结为伪代码的形式。其中上述的讨论对应图 7 (a)中的第 2 到第 5 行。向下移动顺时针珠子和移动逆时针珠子的原理与我们上面讨论的向上移动顺时针珠子类似，不同的只是所涉及的转向。

3.4 基于算盘转向模型的可重构路由算法

基于算盘转向模型和珠子传递原子操作使可重构路由的设计被简化为确定珠子移动方向的规则。

3.4.1 掰腕子路由算法

根据算盘转向模型，四个非关键转向：西向北、北向东、西向南以及南向东是时刻被允许的。而另外四个转向：东向南、南向西、东向北以及北向西则需要通过珠子传递算法进行重构。这四个转向可以分为两组：顺时针组（东向南和南向西）以及逆时针组（东向北和北向西）。一般来说移动珠子意味着在旧的珠子拥有者处禁止一个转向，同时在新的珠子拥有者处打开另外一个属于同一组的转向。因此对不同转向的需求程度自然成为决定珠子移动方

向的依据。为了记录对不同转向的需求，每个节点需要添加三个寄存器：CT_{xy}、$CT_{xy\text{-}n}$和$CT_{xy\text{-}s}$，用于分别表示当前节点、北邻居和南邻居对转向 ***xy*** 的需求，其中

$$xy \in \{es, sw, en, nw\}$$

（***e***：东；***s*** 南；***n***：北；***w***：西； ***es***：东向南，***sw***：南向西……，依此类推）

有了节点对不同转向的需求之后，上下移动珠子可以类比为在一面垂直的墙上上下推动一个质量为 0 的方块。如图 8 所示，珠子被类比成质量为 0 的方块。不失一般性，假设此方块为顺时针珠子。

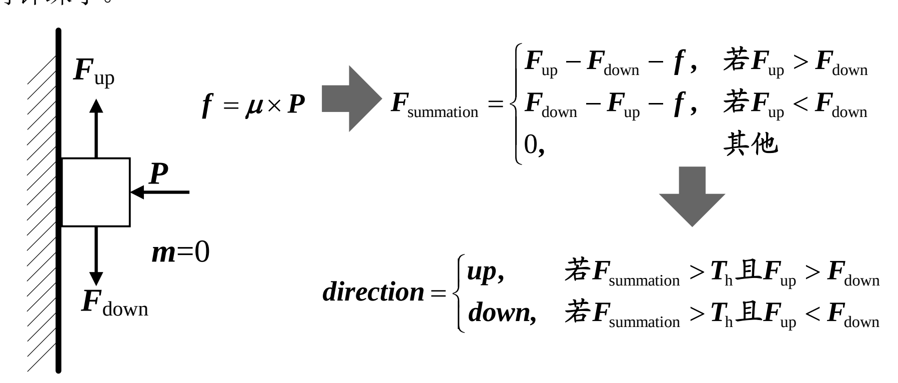


图8. 上下移动质量为 0 的方块示例

在掰腕子算法（arm-wrestling）中，为了将方块向上移动，当前珠子持有者的北邻居向方块施加一个向上的力 F_{up}。这个力实际上反映的是北邻居对东向南转向的需求程度，因为只有把珠子拉上来，该节点才能打开东向南转向。因此北邻居对东向南转向的需求越大，向上的力越大。同时珠子持有者的南邻居为了打开南向西转向也会向方块施加一个力。这个力反映了南邻居对南向西转向的需求程度。对于当前的珠子持有者，一旦它失去珠子便需要禁止相应的转向。例如珠子向上移动，则当前的珠子持有者便会位于新的珠子持有者的下方，需要禁止南向西转向。如果珠子向下移动，则当前珠子持有者需要禁止东向南转向。为了阻止珠子被其他节点抢走，当前的持有者会对珠子的移动施加一个阻力。为了阻止珠子向上移动，这个阻力的方向向下并且反映当前持有者对南向西转向的需求。为了阻止珠子向下移动，阻力的方向向上并且反映当前持有者对东向南转向的需求。向上的力、向下的力以及当前节点的阻力一起形成一个合力，这个合力将决定珠子的移动方向。另外为了防止珠子的抖动，我们设置一个阈值（T_h），只有当最终的合力大于这个阈值时珠子才会被移动。对于移动逆时针珠子，向上的力反映了北邻居对北向西转向的需求，向下的力反映了南邻居对东向北转向的需求。

3.4.2 拔河路由算法

在上述的掰腕子算法中，只有三个节点参与决定珠子移动的方向：珠子的当前持有者、北邻居和南邻居，与珠子持有者距离更远的节点对转向的需求则没有被考虑。为了解决这个问题，拔河算法（tug-war）将位于珠子当前持有者上方和下方的节点分成两个小组。为了向上移动顺时针珠子，持有者上方的所有节点对东向南转向的总需求被视为向上的力 F_{up}。为了将顺时针珠子向下拉动，持有者下方的所有节点对南向西转向的总需求则被视为向下的力 F_{down}。与掰腕子算法相同，珠子的当前持有者同样需要施加阻力避免珠子的移动。另外为了强调与珠子更近的节点的重要性，在传递转向需求过程中，“需求”每经过一个节点即被除以 2。

3.5 实验评估

本节通过与当前常见的路由算法相比较来评估基于算盘转向模型的可重构路由算法。算盘转向模型提供一个“安全”的方式完成无死锁路由的动态生成和动态重构。其中，“安全”表示在算法生成和重构的过程中不会产生死锁。因此我们在挑选参考路由算法时同样选择那些为了解决“安全”问题的算法。比较的路由算法包括一个确定型路由算法：***xy*** 路由；两个部分自适应路由算法：西最先（west-first）[19]和奇偶（odd-even）[20]；一个最短路径完全自适应路由算法[15]。最近提出的路由算法，例如 CQR[30]，O1Turn[31]，和 RCA[32]均是为了解决流量平衡问题，因此并没有参与比较。

自适应路由算法可能产生两个备选输出端口。在这种情况下，选择拥有更多可用缓冲空间（credit）的输出端口。每个路由器包含 5 个输入和输出端口。除了引文[15]，其他文献中路由器均假设每个虚拟网络包含一条虚通道，以及每条虚通道包含一个深度为 4 的输入缓冲队列。而在引文[15]，每个虚拟网络内包含 2 条虚通道，所以为了与其它文献结果公平比较，每个虚通道只分配一个深度为 2 的输入缓冲队列。另外引文[15]不允许一个虚通道被重新分配，除非上一个数据包的尾流控单元已经离开这个虚通道。而对其他路由算法来说，只要一个虚通道收到上一个数据包的尾流控单元，无论该流控单元是否离开这个虚通道都可以重新分配。

本小节首先针对可综合的流量对路由算法的性能进行评估。可综合的流量包括：uniform（均匀）， transpose（对调 ），和 hotspot（热点）。仿真首先假设一个 4×4 的网状网网络，随后为了展现路由算法的可扩展性，同样的仿真又在一个 8×8 的网状网网络中实施。所有的路由算法均在一个时钟周期精确的开源模拟器 Garnet[33]内实现。Garnet 提供两种仿真模式：灵活的和细节的。在本节的仿真中采用的是细节的仿真方式，因为细节的仿真模型提供了修改路由结构的可能。在这个实验中，每个路由器实现一个虚拟网络。

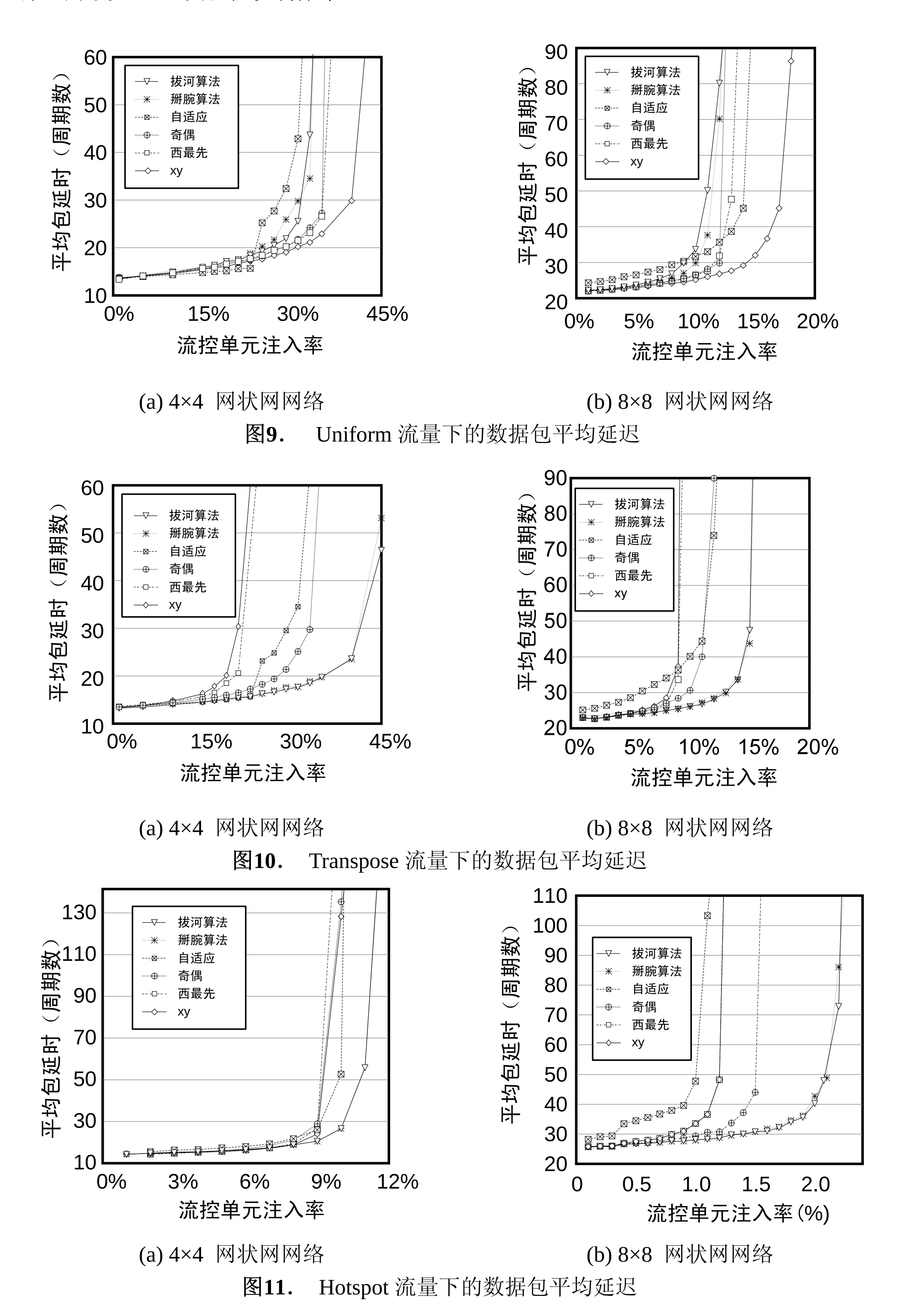
随后，本小节又通过踪迹驱动（trace driven）的仿真方法针对应用程序的流量对路由算法进行评估。应用程序的踪迹是通过开源的全系统模拟器 GEMS[34]获得的。GEMS 在一个商业的全系统模拟器 Simics[35]的基础上增加了对存储模块和处理器模块时序的模拟，即增加了 Ruby 和 Opal 模块。在我们的实验中只加载了 Ruby 模块，并且在 Ruby 模块中选择使用 Garnet 网络。实验所采用的缓存（cache）一致性协议为 MSI_MOSI_CMP_directory。这个协议需要 5 个虚拟网络用于解决协议死锁。在这 5 个虚拟网络中 2 个网络要求数据包按序到达。在这个实验中，为了减少仿真时间，我们采用的是 4×4 的网状网网络。基准测试集为 splash-2[36]和 PARSEC[37]。

3.5.1 可综合流量下的网络性能

在 uniform 流量下，每个节点以相同的概率向其他节点发送数据包。针对 4×4 和 8×8 网状网网络的仿真结果如图 9 所示。水平坐标轴表示流控单元的注入率，垂直坐标轴代表数据包的平均延迟（单位为路由器时钟周期数）。因为 uniform 流量本身的平衡性，确定性路由算法一般会比自适应路由算法获得更好的性能。造成这个现象的主要原因是自适应路由算法经常基于局部信息来做路由选择。这种短视的策略通常会造成网络性能的下降。

如图 9 (a)所示，xy 路由算法获得的性能最好，两个部分自适应路由算法的网络性能类似。基于算盘转向模型的可重构路由算法的性能此时不如部分自适应路由算法。这主要是因为可重构路由算法总是基于历史信息对路由进行重构。但是这种重构策略在 uniform 流量情况下常常是错误的。例如假设东北方向的数据包在当前时间段内具有最高的负载，所以可重构路由算法会在下一个时间段内为东北方向的数据包分配更多的路由自适应度。然而根据 uniform 流量的定义，在下一个时间段内东北方向的数据包的数量很大程度上会非常少。最

短路径完全自适应路由算法在本实验中获得的性能最差。这主要有两个原因：1）其输入缓冲空间最少，2）其流控机制保守。



(a) 4×4 网状网网络　　(b) 8×8 网状网网络

图9.　Uniform 流量下的数据包平均延迟

(a) 4×4 网状网网络　　(b) 8×8 网状网网络

图10.　Transpose 流量下的数据包平均延迟

(a) 4×4 网状网网络　　(b) 8×8 网状网网络

图11.　Hotspot 流量下的数据包平均延迟

为了证明算法的可扩展性，我们又针对 8×8 网状网网络中进行了同样的实验。仿真结果与在 4×4 网状网网络中的结果类似。但是在这个实验中完全自适应路由算法的相对性能

获得了提升。这主要是因为网络中发生冲突的概率随着网络规模的增大而增加。当发生冲突时文献[15]要求数据包在逃避通道内等待。因为逃避通道采用 xy 路由算法，所以此时的完全自适应路由算法也可以像 xy 路由那样获得 uniform 流量均衡的好处。

在真实的世界中，大多数应用程序产生非均匀的流量，例如 transpose 流量。在 transpose 流量中，源节点 s 总是向目的节点 d 发送数据包，其中 $\boldsymbol{d_i} = \boldsymbol{s}_{(\boldsymbol{i}+\boldsymbol{b}/2) \bmod \boldsymbol{b}}$，$\boldsymbol{b}$ 是用于索引所有节点需要的编号长度。由图 10 (a)可以看出在 transpose 流量情况下，基于算盘转向模型的可重构路由算法获得最好的网络性能，因为它们可以为所有的数据包提供完全自适应度。Transpose 流量很容易造成严重的网络拥塞，因此 xy 路由算法此时获得性能最差。西最先路由算法可以获得比 xy 路由算法更好的性能，因为它可以为向东传输的数据包提供完全的自适应度。但是其性能的提高是有限的，因为向西传输的数据包仍然遭受着严重的网络拥塞。奇偶路由可以为各个方向的数据包提供相对均衡的自适应度，所以它可以获得比“西最先”更好的网络性能。但是因为其提供的自适应度是不完全的，所以无法获得与基于算盘转向模型的可重构路由算法同样高的性能。完全自适应路由算法同样可以提供完全的自适应度，但是其缓冲区较少以及流控机制保守，因此获得的性能同样低于基于算盘转向模型的可重构路由算法。如图 10 (b)展示了在 8×8 网状网网络中进行同样实验的仿真结果。虽然各个路由算法之间的相对性能没有改变，但是我们提出的路由算法与当前路由算法之间的性能差距拉大了。这主要是因为我们提出的算法能更好地处理在大规模网络中的拥塞问题。

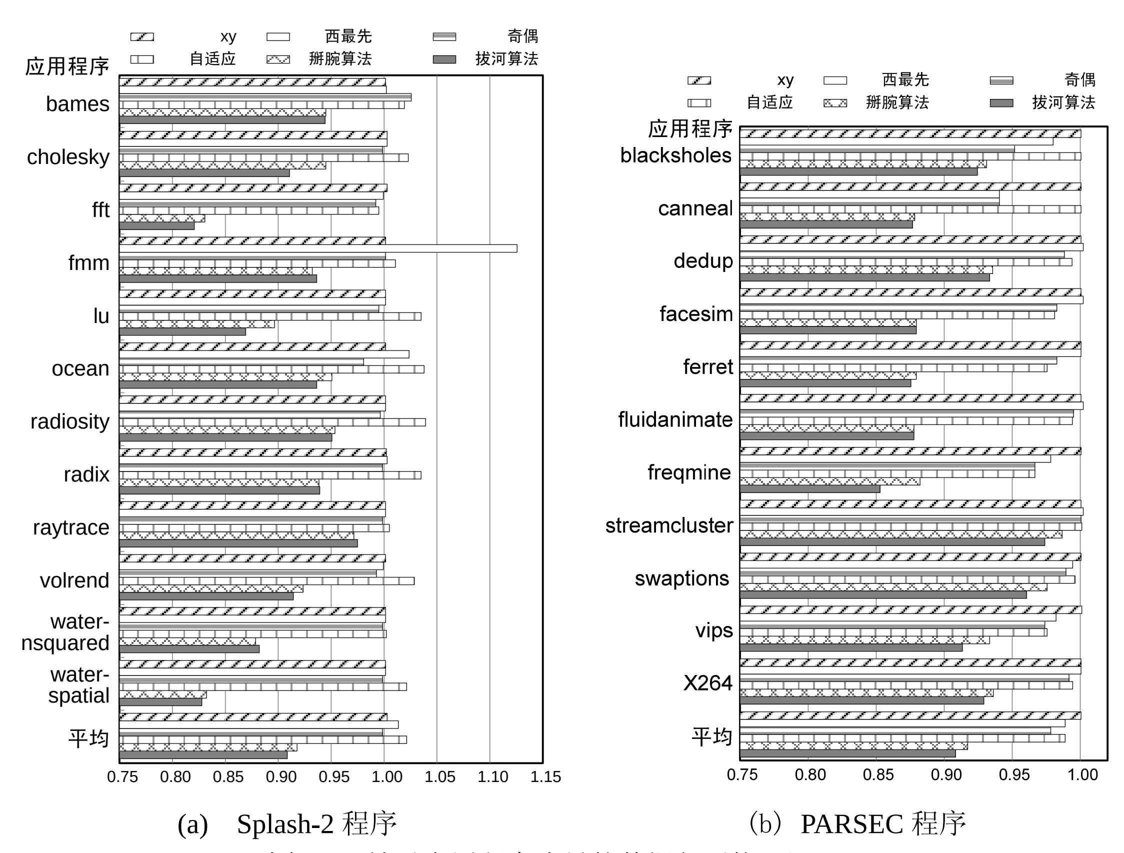
应用程序中的突发流量容易造成网络热点，从而加剧网络的拥塞问题。在接下来的两个仿真假设存在 4 个热点：节点 0，节点 4，节点 8，和节点 12。这 4 个热点相对其他节点有额外 20%的机会获得数据包。我们选择这 4 个节点用于模拟 4 个存储控制器被频繁访问的情况。在这种情况下，西向数据包非常容易产生拥塞。如图 11 (a)所示的仿真结果表明基于算盘转向模型的路由算法可以获得最好的性能。这是因为它可以为所有的数据包提供完全的自适应度。同样完全自适应路由算法因为具有较少的缓存空间以及使用保守的流控机制，所以其性能不如基于算盘转向模型的可重构路由算法，甚至不如奇偶路由算法。西最先和 xy 路由因为不能提供自适应度，所以性能最差。当网络规模扩大后，网络的拥塞问题会更严重。如图 11 (b)所示，此时基于算盘转向模型的可重构路由算法的性能优势更加明显。

3.5.2 针对应用程序流量的网络性能

在图 12 (a)中，所有路由算法在 splash-2 基准测试集的网络平均延迟全部以 xy 路由的网络延迟为基准归一化。基于算盘转向模型的掰腕子算法和拔河算法通过动态地为突发流量分配更多的自适应度的方式大幅降低了网络的平均延迟。总的来说，对所有的测试基准集内的应用程序，我们提出的 2 个算法都有不同程度的性能提升。但是对于不同类型的应用程序，性能提升的程度不同。例如，对于那些像快速傅里叶变换和水分子力和势能分布（water-spatial） 等容易产生冲突的应用程序，网络性能的提升是比较显著的。然而对于类似光线追踪（raytrace）和洋流模拟（ocean）这些低冲突的应用程序，性能的提升则相对较小。例如，洋流模拟所产生的 60%以上的流量是本地流量，此时掰腕子算法和拔河算法带来的性能提升分别为 6%和 7%。总的来说，对于 splash-2 基准测试集我们提出的算法平均可以获得 10%的网络性能提升，最大可以获得 19%的性能提升。

为了增加上述实验的可信度，我们在 PARSEC 基准测试集下重做了上述实验。仿真结果如图 12 (b)所示，图中网络的平均数据包延迟同样以 xy 路由的网络延迟为基准归一化。与上述的实验结果相同，基于算盘转向模型的可重构路由算法在 PARSEC 基准测试集下同样可以大幅地提高网络性能。特别对于类似 canneal（用于优化芯片内布局布线的缓存感知

退火算法）和 freqmine（频繁项集挖掘）等高度拥塞的应用程序，网络性能的提升尤其可观。例如对于 freqmine 应用程序，拔河算法可以减少 15%的平均延迟，而掰腕子算法可以减少 12%。但是对于类似 streamcluster（输入流的在线集聚算法）等低冲突的应用程序，网络性能的提升则相对较少。例如对于 streamcluster，掰腕子和拔河算法分别减少 2%和 3%的数据包平均延迟。总的来说，针对 PARSEC 基准测试集，我们提出的可重构路由算法最多可以减少 15%平均可以减少 10%的数据包访问延迟。



(a)　Splash-2 程序　　（b）PARSEC 程序

图12.　针对应用程序流量的数据包平均延迟

4　总结

本文介绍了三种利用应用程序流量信息的可重构路由算法。其中两种为离线可重构算法，一种为在线可重构算法。离线算法利用事先采集的应用程序的通信特征来确定路由规则，从而达到平衡网络流量、提升网络性能的目标。这种方式的好处是可以利用全局的流量信息来对系统进行优化，但是却只能在应用程序定制的系统中使用。在通用处理器系统中，由于不能提前采集程序的流量特性，所以只能使用在线的可重构路由算法。基于算盘转向模型的可重构路由算法可以通过动态地改变每个节点禁止的转向来实现路由自适应度的按需分配。由于高负载方向的数据包可以动态地获得更多的路由自适应度用于平衡网络流量，所以网络性能可以获得较大提升。针对非均匀可综合流量，Splash-2 和 PARSEC 基准测试程序集的仿真实验表明，相对已有路由算法，基于算盘转向模型的可重构路由算法在流量模型下可获得较明显的性能提升。

参考文献：

[1]　D. Geer, “Chip makers turn to multicore processors”, *Computer*, vol. 38, no. 5, pp: 11-13, May, 2003.

[2]　W.J. Dally and B. Towles, “Route packets, not wires: on-chip interconnection networks”, in Proceedings of *Design Automation Conference*, pp: 684-689, 2001.

[3] W.J. Dally and B. Towles, “Principles and practices of interconnection networks”, *Morgan Kaufmann*, 2004.

[4] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Lee Jae-Wook, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, “The Raw microprocessor: a computational fabric for software circuits and general-purpose programs”, *IEEE Micro*, vol. 22, no. 2, pp: 25-35, 2002.

[5] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, S. Borkar, “An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS”, *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp: 29-41, Jan. 2008.

[6] D.R. Fan, N. Yuan, J. C. Zhang, Y. B. Zhou, W. Lin, F.L. Song, X. C. Ye, H. Huang, L. Yu, G. P. Long, H. Zhang, and L. Liu, “Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions”, *Journal of Computer Science and Technology*, vol. 24, no. 6, 2009.

[7] Jongman Kim, Dongkook Park, T. Theocharides, N. Vijaykrishnan, C.R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in Proceedings of *Design Automation Conference*, pp. 559- 564, 2005.

[8] A. Singh, W.J. Dally, A.K. Gupta, B. Towles, "GOAL: a load-balanced adaptive routing algorithm for torus networks," in Proceedings of *International Symposium on Computer Architecture*, pp. 194- 205, 2003.

[9] J. Brand, C. Ciordas, K. Goossens, T. Basten, "Congestion-Controlled Best-Effort Communication for Networks-on-Chip," *Design, Automation & Test in Europe Conference & Exhibition*, pp.1-6, 2007.

[10] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, T. Nachiondo, "A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks," *International Symposium on High-Performance Computer Architecture*, pp. 108- 119, 2005.

[11] P. Gratz, B. Grot, S.W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in Proceedings of *High Performance Computer Architecture*, pp.203-214, 2008.

[12] D. Park, R. Das, C. Nicopoulos, J. Kim, N. Vijaykrishnan, R. Iyer, C.R. Das, "Design of a Dynamic Priority-Based Fast Path Architecture for On-Chip Interconnects," in Proceedings of *IEEE Symposium on High-Performance Interconnects*, pp.15-20, 2007.

[13] S.A. Felperin, L. Gravano, G.D. Pifarre, J. Sanz, "Fully-adaptive routing: packet switching performance and wormhole algorithms," in Proceedings of *the ACM/IEEE Conference on Supercomputing*, pp.654-663, 1991.

[14] W.J. Dally, H. Aoki, “Deadlock-free adaptive routing in multicomputer networks using virtual channels”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 4, pp: 466-475, Apr. 1993.

[15] J. Duato, “A new theory of deadlock-free adaptive routing in wormhole networks”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, Dec. 1993.

[16] Y. M. Boura, C.R. Das, “Efficient fully adaptive wormhole routing in n-dimensional meshes”, in Proceedings of *International Conference on Distributed Computing Systems*, pp: 589-596, 1994.

[17] J.H. Upadhyay, V. Varavithya, P. Mohapatra, “Efficient and balanced adaptive routing in two-dimensional meshes”, in Proceedings of *IEEE Symposium on High-Performance Computer Architecture*, pp: 112 -121, 1995.

[18] L.S. Peh and W.J. Dally, “A delay model and speculative architecture for pipelined routers”, in Proceedings of *International Symposium on High-Performance Computer Architecture*, pp: 255 -266, 2001.

[19] C.J. Glass and L.M. Ni, “The Turn Model for Adaptive Routing”, in Proceedings of *International Symposium on Computer Architecture*, pp: 278-287, 1992.

[20] G.M. Chiu, "The odd-even turn model for adaptive routing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp: 729-738, Jul. 2000.

[21] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "Application Specific Routing Algorithms for Networks on Chip", *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 3, Mar. 2009.

[22] M.A. Kinsy, M.H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas, "Application-aware deadlock-free oblivious routing", in Proceedings of *international symposium on Computer architecture (ISCA'09)*, pp: 208—219, 2009.

[23] B. Fu, Y. Han, J. Ma, H. Li, and X. Li, "An abacus turn model for time/space-efficient reconfigurable routing", in Proceedings of *international symposium on Computer architecture (ISCA'11)*, pp: 259-270, 2011.

[24] T.L. Rodeheffer and M.D. Schroeder, "Automatic reconfiguration in Autonet", in Proceedings of *ACM symposium on Operating systems principles*, pp: 183-197, 1991.

[25] O. Lysne and J. Duato, "Fast dynamic reconfiguration in irregular networks", in Proceedings of *International Conference on Parallel Processing*, pp: 449-458, 2000.

[26] R. Casado, A. Bermudez, F.J. Quiles, J.L. Sanchez, J. Duato, "Performance evaluation of dynamic reconfiguration in high-speed local area networks", in Proceedings of *International Symposium on High-Performance Computer Architecture*, pp: 85-96, 2000.

[27] R. Casado, A. Bermudez, J. Duato, F.J. Quiles, J.L. Sanchez, "A protocol for deadlock-free dynamic reconfiguration in high-speed local area networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no.2, pp: 115-132, Feb. 2001.

[28] D. Avresky and N. Natchev, "Dynamic reconfiguration in computer clusters with irregular topologies in the presence of multiple node and link failures", *IEEE Transactions on Computers*, vol. 54, no. 5, pp: 603-615, May 2005.

[29] O. Lysne, J.M. Montanana, J. Flich, J. Duato, T. M. Pinkston, T. Skeie, "An efficient and deadlock-free network reconfiguration protocol", *IEEE Transactions on Computers*, vol. 57, no. 6, pp:762-779, June 2008.

[30] A. Singh, W.J. Dally, A.K. Gupta, and B. Towles, "Adaptive channel queue routing on k-ary n-cubes", in Proceedings of *ACM symposium on Parallelism in Algorithms and Architectures*, pp: 11-19, 2004.

[31] D. Seo, A. Ali, W.T. Lim, N. Rafique, and M. Thottethodi, "Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks", in Proceedings of *international symposium on Computer Architecture*, pp: 432-443, 2005.

[32] P. Gratz, B. Grot, S.W. Keckler, "Regional congestion awareness for load balance in networks-on-chip", in Proceedings of *IEEE International Symposium on High Performance Computer Architecture*, pp:203-214, 2008.

[33] N. Agarwal, L.S. Peh, N. Jha, "Garnet: A detailed interconnection network model inside a full-system simulation framework", *TR CE-P08-001, Princeton University*, 2007.

[34] M.M.K. Martin, D.J. Sorin, B.M. Beckmann, M.R. Marty, M. Xu, A.R. Alameldeen, K.E. Moore, M.D. Hill, D.A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset", *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp: 92-99, 2005.

[35] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform", *Computer*, vol. 35, no.2, pp: 50-58, 2002.

[36] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta, "The SPLASH-2 programs: characterization and methodological considerations", in Proceedings of *International Symposium on Computer Architecture*, pp:24-36, 1995.

（下转第 27 页）